- Ridge Regression
- Least Squares
- Regression
- Logistic Function View
- NoIV's Theorem
- Perceptron Algorithm
- Online vs. Batch implementations

Online Learning:

1. Online Learning Scenarios:
   - Ideally, you want to update $h(t)$ as we get new data.
   - e.g. stock market forecasting.

2. Given a sequence of examples $(x_1, y_1), (x_2, y_2), \ldots$
   - This is a one-time deal (i.e., $f_1, f_2, \ldots$)

3. Given a set of examples $(x_i, y_i), i = 1 \ldots m$
   - Learn $f: \mathcal{X} \rightarrow \mathcal{Y}$ such that it performs well on new set $\mathcal{E}(x_i, Y)$. 
   - What were discussed so far are batch learning scenarios?

Online Learning:
Online learning, but batch learning may be harder. Some amount of data, batch learning uses. In this one, the more data, the more accurate. 

Online learning can deal with infinitely large datasets. Online algorithms are usually easier to implement.

Online learning applied to batch scenario has some benefits:

1. Algorithm can be applied to batch scenario.
2. Learning can be applied to online scenario.

\[
\begin{align*}
\text{Online learning} & \quad \text{Learning} \\
\text{(batch learning)} & \quad \text{(online learning)} \\
\text{(batch learning)} & \quad \text{(online learning)} \\
\text{(batch learning)} & \quad \text{(online learning)} \\
\end{align*}
\]
Suppose the current classifier makes a mistake on x. At $x$, 

where $S(x) = \text{sign}(wx+b)$.

The sign function is a linear classifier parameterized by weights $w$ and $b$.

Perceptron is a linear classifier algorithm.

Let $x \in \mathbb{R}^d$, $y \in \{-1, 1\}$ binary classification task.

We assume a batch scenario. Training data $(x_i, y_i)_{i=1}^m$.

Perceptron Algorithm (an example of online (incremental)
Let $n = \sum_{i=2}^{1} \frac{\frac{1}{2}^i}{1 + \frac{1}{2}^i}$.

$\Rightarrow$ very effective classification function, overfitting.

$\Rightarrow \leq \leq \leq$ No linear classifier

Separable

Linearly separable?

Does perceptron also behave when data is not linearly separable?

In this theorem, the perceptron algorithm will converge.

(We refer to the algorithm convergence.)

Theorem: if your data is linearly separable.

Training Set: (2, y will the algorithm converge)

If we keep our distance, the perceptron algorithm will converge.

"If we keep our distance, the perceptron algorithm will converge."
data, we approach it by one example at a time.

Gradient: Rather than computing gradient on entire
Perceptron update is an instance of "stochastic approxi-
\[ \Delta w = \nabla f(x) \]
\[ w = w + \Delta w \]
f the gradient update to Line 5 of the perceptron:

3. Let \( r = 1 \) (constant step size), then note the similari-
where \( r \) is step size.

\[ x_{\text{new}} = x_{\text{old}} + r \nabla f(x) \]
\[ w = w + r \Delta w \]

Thus, to minimize \( L(w) \), we use gradient descent.

\[ \Delta L(w) = \frac{\partial L}{\partial w}(x) \]

2. Take derivative of \( L(w) \) w.r.t. \( w \).

\[ \text{examine to hyperplane for } \]
\[ \text{of incorrect examples (minimum distance from moment} \]
\[ \text{want to such that it minimizes the dot product of } \]
\[ \text{set of misclassifed examples} \]

\[ \text{in } \]}
\[ \text{loss}(w) = \frac{1}{2} ||y-w||^2 \]

An alternative view of the perceptron via the loss function.
\[ z = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \]

\[ \text{Gradient: } \Delta L = \frac{\partial}{\partial w} J(w, b) = X^T (y - Xw) \]

\[ \text{Loss Function: Mean Square Error} \]

\[ J(w, b) = \frac{1}{m} \sum_{i=1}^{m} (y_i - (wx_i + b))^2 \]

\[ y = wx + b \]

\[ f(x) = \frac{1}{2} \left[ \begin{array}{c} b \\ w \end{array} \right] ^\top \left[ \begin{array}{c} x \\ w \end{array} \right] \]

\[ \text{Linear Regression: Least Squares} \]
The residual vector $y - X\omega$ is orthogonal to $X$. From gradient we get $X^T(y - X\omega) = 0$. Although closed-form solution for $\omega$ is available, we may opt to get $\omega$ by gradient descent.

Although closed-form solution for $\omega$ is available, we may opt to get $\omega$ by gradient descent. We derive $\omega$ by iterating by descent.

\[
(X^TX)^{-1}X^Ty = \omega
\]

We view $X$ as column vectors instead:

\[
[X_1 X_2 \cdots X_n] X = X_1 \omega_1 + X_2 \omega_2 + \cdots + X_n \omega_n
\]

The generic view of $\omega$ is:

\[
\omega = \frac{1}{m} \sum_{i=1}^{m} (y_i - X_i \omega) X_i^T
\]
A basic regression:

\[ y = \beta_0 + \beta_1 x + \epsilon \]

\[ \text{Ridge Regression: } \hat{\beta} = \arg\min_{\beta} (y - X\beta)^2 + \lambda \beta^T \beta \]

\[ \lambda = \frac{1}{2} \sum \frac{w_i}{\sqrt{\sum (y_i - \hat{y}_i)^2}} \]

\[ \lambda \text{ should be tuned via cross-validation} \]

\[ \text{Normalization via SVD} \]

\[ \min_{\| \beta \|_2 = 1} || y - X\beta ||^2 \]

\[ \text{Ridge Regression: } \hat{\beta} = \arg\min_{\beta} (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta \]

\[ \text{Note: } \lambda > 0 \text{, the inverse is no longer even if } X^T X \text{ is not full rank.} \]

\[ L_{\text{ridge}}(\beta) = \frac{1}{2} (X^T X + \lambda I)^{-1} X \hat{y} \]

\[ \text{Solution: } \hat{\beta} \text{ is } \frac{1}{n} (X^T X + \lambda I)^{-1} X^T y \]

\[ \text{Note: } \lambda > 0, \text{ the inverse is no longer even if } X^T X \text{ is not full rank.} \]

\[ \text{Minimizing } L_{\text{ridge}}(\beta) \text{ would yield } \beta \text{ penalized with } \lambda \text{ norm.} \]

\[ \text{Ridge Regression: } \hat{\beta} = \arg\min_{\beta} (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta \]

\[ \text{Some coefficients in } \beta \text{ are zero.} \]

\[ \text{Well trade-off bias for variance by shrinking.} \]

\[ \text{Least squares estimates may have high variance.} \]

\[ \text{Ridge Regression } \hat{\beta} \text{ Shrinkage} \]